Amendments to the Claims:

This listing of claims will replace all prior versions and listings of claims in the above-identified application.

Listing of Claims:

1. (Previously presented) A method of verifying program code conversion performed by an emulator, comprising:

a) dividing a subject code into a plurality of blocks and executing one of the blocks of subject code through an emulator in a process image on a subject processor according to an emulation context up until a comparable point in the subject code to provide an emulated machine state;

b) performing a context switch to a native context and executing the same block of subject code natively in the same process image on the same subject processor up until the same comparable point in the subject code to provide a native machine state; and

c) comparing the native machine state from execution of the one block of subject code natively on the subject processor against the emulated machine state from execution of the same block of subject code on the same subject processor through the emulator at the comparable point in the subject code;

wherein the native machine state includes a native memory image and the emulated machine state includes an emulated memory image and

(a) and/or (b) includes selectively isolating access to a memory associated with the subject processor to obtain the native memory image and/or the emulated memory image, respectively.

2. (Canceled)

3. (Previously presented) The method of claim 1, comprising performing (a) prior to performing (b).

4. (Previously presented) The method of claim 3, wherein:

(a) further comprises providing an emulated image of the subject processor;

(b) further comprises providing a native image of the subject processor following the native execution of the program code; and

(c) further comprises comparing the emulated image of the subject processor against the native image of the subject processor.

5. (Previously presented) The method of claim 4, wherein (a) comprises providing the emulated image of the memory in a load/store buffer associated with the memory, such that the memory is not affected by executing the subject code through the emulator.

6. (Original) The method of claim 5, wherein the emulated image of the subject processor includes an image of one or more registers.

7. (Original) The method of claim 6, wherein the emulated image of the subject processor includes an image of one or more condition code flags.

8-9. (Canceled)

10. (Previously presented) The method of claim 1, further comprising selectively switching between the emulation context for running the emulator on the subject processor, a target execution context for executing target code produced by the emulator on the subject processor, and the native context where the subject code runs natively in the subject processor.

11. (Previously presented) The method of claim 10, wherein both the native context and the emulation context employ a single image of the subject code.

12. (Canceled)

13. (Previously presented) The method of claim 1, comprising selecting between two or more verification modes, and dividing the subject code into the plurality of blocks according to the selected verification mode.

14. (Previously presented) The method of claim 13, comprising repeating the executing and comparing for each of the plurality of blocks.

15. (Previously presented) The method of claim 14, wherein in a first verification mode each block comprises a single instruction of subject code; in a second verification mode each block comprises a basic block comprising a sequence of instructions from a unique entry instruction to a unique exit instruction; and in a third verification mode each block comprises a group block comprising a plurality of the basic blocks.

16. (Previously presented) The method of claim 1, comprising:

dividing a large segment of the subject code into a plurality of smaller blocks, each block containing one or more instructions from the large segment of subject code; and

performing a verification comparison at a block boundary between each pair of consecutive neighbouring blocks in the plurality of blocks.

17. (Previously presented) The method of claim 16, comprising:

providing the subject processor in the emulation context, where control of the processor rests with the emulator, performing program code conversion on a current block BBn to produce a corresponding block of converted target code, and patching an immediately preceding block of subject code BBn-1 with a return jump;

executing a context switch routine to enter the native context, and executing the immediately preceding block of subject code BBn-1 natively by the subject processor, such that the executing terminates with the return jump;

executing a context switch routine to return to the emulation context, and performing the verification comparison by comparing a native machine state representing the subject processor following execution of the immediately preceding block BBn-1 with an emulated machine state representing a virtual model of the subject processor held by the emulator following execution of the immediately preceding block BBn-1;

executing a context switch to a target execution context, and modelling execution of the target code corresponding to the current block of subject code BBn in the virtual model of the subject processor held by the emulator, thereby leaving the virtual model in a machine state representing the end of the current block BBn; and

repeating the above for each subsequent block in the plurality of blocks, unless the verification comparison reveals an error in the program code conversion.

18. (Original) The method of claim 17, further comprising restoring the immediately preceding block BBn-1 to remove the return jump.

19. (Currently amended) The method of claim 1, further comprising:

selecting a block of the subject code;

executing the block of subject code on the subject processor through the emulator; and

appending a return jump to the block of subject code, and executing the block of subject code natively on the subject processor terminating with the return jump, such that the return jump returns control of the processor to the emulator.

20-99. (Canceled)

100. (New) A system of verifying program code conversion performed by an emulator, comprising:

a processor;

a memory, coupled to the processor; and

logic, stored on the memory and executed on the processor, for:

dividing a subject code into a plurality of blocks and executing one of the blocks of subject code through an emulator in a process image on the processor according to an emulation context up until a comparable point in the subject code to provide an emulated machine state;

performing a context switch to a native context and executing the same block of subject code natively in the same process

image on the processor up until the same comparable point in the subject code to provide a native machine state; and

comparing the native machine state from execution of the one block of subject code natively on the processor against the emulated machine state from execution of the same block of subject code on the processor through the emulator at the comparable point in the subject code;

wherein the native machine state includes a native memory image and the emulated machine state includes an emulated memory image and the dividing a subject code and/or the performing a context switch includes selectively isolating access to a memory associated with the processor to obtain the native memory image and/or the emulated memory image, respectively.

101. (Previously presented) The system of claim 100, wherein:

the logic for dividing the subject code comprises logic for providing an emulated image of the subject processor;

the logic for performing a context switch comprises logic for providing a native image of the subject processor following the native execution of the program code; and

the logic for comparing the native machine state comprises logic for comparing the emulated image of the subject processor against the native image of the subject processor.

102. (Previously presented) The system of claim 100, the logic further comprising logic for:

selecting between two or more verification modes, and

dividing the subject code into the plurality of blocks according to the selected verification mode.

103. (Previously presented) The system, of claim 102, wherein in a first verification mode each block comprises a single instruction of subject code; in a second verification mode each block comprises a basic block comprising a sequence of instructions from a unique entry instruction to a unique exit instruction; and in a third verification mode each block comprises a group block comprising a plurality of the basic blocks.

104. (Previously presented) The system of claim 100, comprising the logic further comprising logic for:

dividing a large segment of the subject code into a plurality of smaller blocks, each block containing one or more instructions from the large segment of subject code; and

performing a verification comparison at a block boundary between each pair of consecutive neighboring blocks in the plurality of blocks.

105. (Previously presented) The system of claim 100, the logic further comprising logic for:

selecting a block of the subject code;

executing the block of subject code on the subject processor through the emulator; and

appending a return jump to the block of subject code, and executing the block of subject code natively on the subject processor terminating with the return jump, such that the return jump returns control of the processor to the emulator.

106. (Previously presented) A computer programming product of verifying program code conversion performed by an emulator, comprising:

a physical memory; and

logic, stored on the memory fro execution on a processor, for:

dividing a subject code into a plurality of blocks and executing one of the blocks of subject code through an emulator in a process image on the processor according to an emulation context up until a comparable point in the subject code to provide an emulated machine state;

performing a context switch to a native context and executing the same block of subject code natively in the same process image on the processor up until the same comparable point in the subject code to provide a native machine state; and

comparing the native machine state from execution of the one block of subject code natively on the processor against the emulated machine state from execution of the same block of subject code on the processor through the emulator at the comparable point in the subject code;

wherein the native machine state includes a native memory image and the emulated machine state includes an emulated memory image and the dividing a subject code and/or the performing a context switch includes selectively isolating access to a memory associated with the processor to obtain the native memory image and/or the emulated memory image, respectively.

107. (Previously presented) The computer programming product of claim 106, wherein:

the logic for dividing the subject code comprises logic for providing an emulated image of the subject processor;

the logic for performing a context switch comprises logic for providing a native image of the subject processor following the native execution of the program code; and

the logic for comparing the native machine state comprises logic for comparing the emulated image of the subject processor against the native image of the subject processor.

108. (Previously presented) The computer programming product of claim 106, the logic further comprising logic for:

selecting between two or more verification modes, and

dividing the subject code into the plurality of blocks according to the selected verification mode.

109. (Previously presented) The computer programming product, of claim 108, wherein in a first verification mode each block comprises a single instruction of subject code; in a second verification mode each block comprises a basic block comprising a sequence of instructions from a unique entry instruction to a unique exit instruction; and in a third verification mode each block comprises a group block comprising a plurality of the basic blocks.

110. (Previously presented) The computer programming product of claim 106, comprising the logic further comprising logic for:

dividing a large segment of the subject code into a plurality of smaller blocks, each block containing one or more instructions from the large segment of subject code; and

performing a verification comparison at a block boundary between each pair of consecutive neighboring blocks in the plurality of blocks.

111. (Previously presented) The computer programming product of claim 106, the logic further comprising logic for:

selecting a block of the subject code;

executing the block of subject code on the subject processor through the emulator; and

appending a return jump to the block of subject code, and executing the block of subject code natively on the subject processor terminating with the return jump, such that the return jump returns control of the processor to the emulator.